Drop The Beat

한양여자대학교

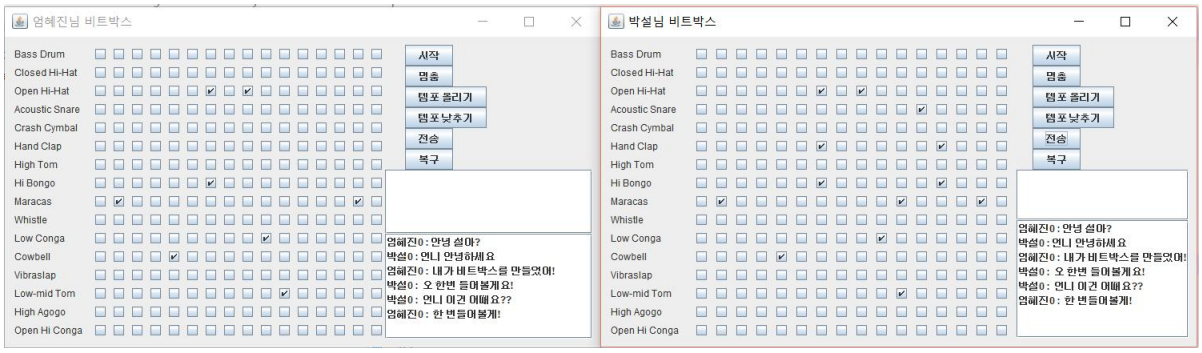
한양여자대학교

객체지향언어 실습

216230105 엄혜진

목차

1. 프로그램 설명



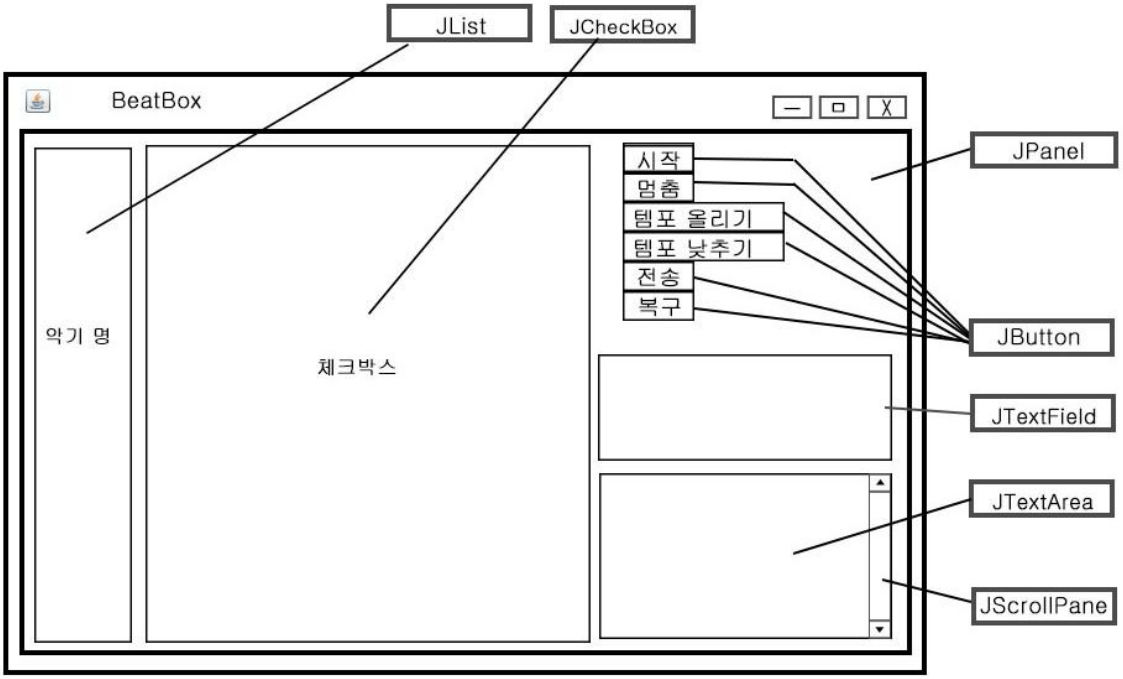
'비트박스' 음악 재생 프로그램은 박자 16개를 나타내는 16개의 체크가 되어있지 않은 상자를 체크하게 되면 여러 소리가 합쳐져 연주되는 프로그램입니다.

1. 체크상자 256개와 악기 명에 해당하는 레이블 16개 마지막으로 버튼 네 개가 들어가있는 GUI 프로그램입니다.
2. 버튼 4개에 대해서는 ActionListener이 등록되어있어 사용자가 시작 버튼을 누를 때까지 기다렸다가 체크상자 256개를 모두 확인하여 그 상태를 알아내고 미디 트랙을 만들어 냅니다.
3. 또한 시퀀서를 받아오고 시퀀스를 만들고 트랙을 만드는 과정을 통해 미디 시스템을 설정하였으며, 시퀀서 메소드인 setLoopCount()를 사용하여 반복시킬 횟수를 설정하였습니다.
4. 빠르기는 시퀀서의 템포를 써서 더 빠르게 또는 더 느리게 조절할 수 있도록 하였으며 루프를 반복할 때마다 새로운 빠르기를 적용할 수 있도록 설정하였습니다.
5. 시작 버튼을 클릭하게 되면 실제 행동이 시작이 되며 멈춤 버튼을 누를 때까지 계속 연주하도록 시퀀서를 시작합니다.
6. 복구 버튼을 누르게 되면 이전 것을 복구할 수 있도록 설정하였습니다.
7. 상대방이 만든 패턴을 비트박스 서버로 보내면 그 패턴을 캡처 할 수 있습니다.
8. 메시지와 패턴이 함께 들어오면 메시지를 클릭해서 그 패턴을 불러올 수 있습니다.

2. 배경기술

2.1 GUI

2.2.1 화면 설계



비트박스 프로그램에는

JFrame, JPanel, JList, JTextField, JButton, JScrollPane, JCheckBox

7가지가 있습니다.

JFrame: 메인 프레임을 말합니다.

JPanel: 보조프레임을 말합니다.

JList: 임의의 객체 유형 배열을 말합니다.

JTextField: 글을 적을 수 있는 텍스트 필드입니다.

JButton: 버튼을 말합니다.

JScrollPane: 스크롤 바를 말합니다.

JCheckBox: 체크박스를 말합니다.

2.1.2 이벤트 처리 설계

2.1.2.1 서버

서버에는 이벤트 처리 코드가 존재하지 않습니다.

2.1.2.2 클라이언트

1. 어떤 이벤트? ActionEvent

- 버튼은 ActionEvent의 소스므로 버튼에는 addActionListener()라는 메소드가 있어서 그 버튼에 관심 있는 객체(리스너)에서 알려줄 수 있습니다.
- 이벤트가 일어나면 버튼에서 클래스로 연락을 해야 하는데, 그 경우에는 리스너 인터페이스에 있는 메소드를 호출합니다.
- ActionListener역할을 할려면 인터페이스에 하나밖에 없는 메소드인 actionPerfomed()를 구현해야 합니다.

1.1 어떤 리스너? ActionEventLisenter

1.2 어떤 메소드? actionPerformed(ActionEvent a){

…

}

1.3 어떤 종류?

```
Start.addActionListener(new MyStartListener());
```

```
Stop.addActionListener(new MyStopListener());
```

```
upTempo.addActionListener(new MyUpTempoListener());
```

```
downTempo.addActionListener(new MyDownTempoListener());
```

```
sendIt.addActionListener(new MySendListener());
```

```
read.addActionLitener(new MyreadListener());
```

2. 어떤 이벤트? ListSelectionEvent

- 현재의 선택 범위에서의 변경을 기술하는 이벤트

2.1 어떤 리스너? ListSelectionListener

2.2 어떤 메소드? valueChanged(ListSelectionEvent le){

…

}

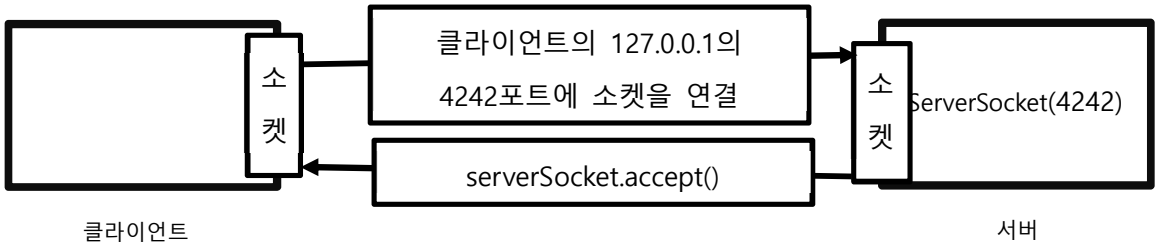
2.3 어떤 종류?

```
incomingList.addListSelectionListener(new MyListSelectionListener());
```

2.2 네트워킹

2.2.1 서버 역할

- 접속 수락

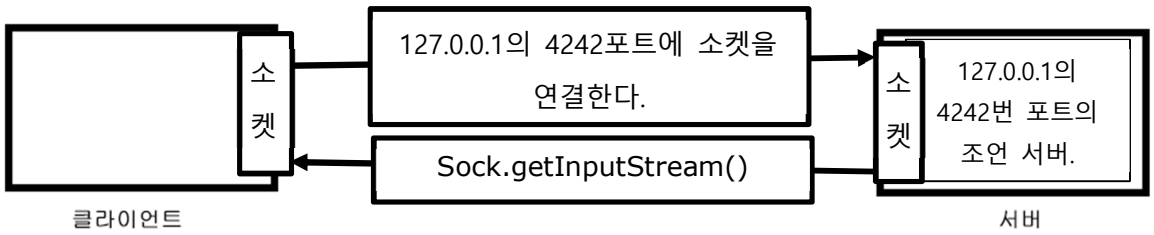



```
// 서버 Socket 포트 번호 4242로 설정
ServerSocket serverSocket = new ServerSocket(4242);
// 서버에서 클라이언트와 통신하기 위한 새로운 Socket을 만든다.
Socket clntSocket = serverSocket.accept();
```

서버에서 클라이언트와 통신하기 위해 새로운 Socket을 만들고, 포트번호를 4242로 지정합니다.
accept() 메소드에서는 클라이언트의 Socket이 연결할 때까지 계속 대기 상태를 유지합니다.

2.2.2 클라이언트 역할

- 연결하기



클라이언트에서 서버 애플리케이션으로 Socket 연결을 합니다.
클라이언트에서는 IP주소와 포트번호를 알고 있습니다.

```
// 서버의 IP주소와 TCP 포트번호
// 서버에 Socket 연결을 한다.
Socket sock = new Socket("127.0.0.1", 4242);
```

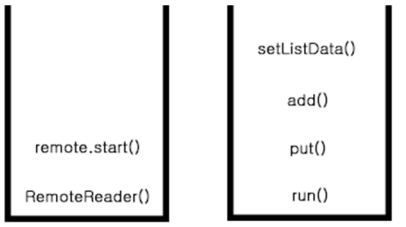
2.3 스레드

2.3.1 서버에서의 용도

우선 클라이언트로부터 데이터를 읽어 들어오기 위해 inputstream을 생성해 줍니다.
서버에서 스레드는 ObjectInputStream 생성자에 클라이언트 socket을 넣어, 클라이언트로부터 데이터를 불러옵니다. 데이터가 생성자의 객체에 접근하면 readObject() 메소드를 사용하여 객체정보를 Object로 반환해줍니다.

2.3.2 클라이언트에서의 용도

일반적으로 스레드는 JVM 스레드 스케줄러가 실행시킬 스레드를 선택했다가 다른 스레드를 기회를 주는 과정을 반복함에 따라 실행 가능한 상태와 실행중인 상태 사이에서 왔다 갔다 합니다.
클라이언트에서 스레드는 메시지를 읽어오는 스레드를 만들고 시작하여 서버로부터 데이터를 읽어옵니다.
메시지가 들어오면 객체 두 개를 읽어오는데 클라이언트에서는 직렬화된 객체 두 개를 읽어옵니다. 그리고 JList 구성요소에 추가하는데 추가하는 과정은 우선 목록 데이터에 대한 Vector 객체를 만들고 JList에 그 Vector를 이용하여 목록에 표시하라는 명령을 하는 작업을 합니다.



<클라이언트 스레드 진행순서>

2.4 컬렉션 프레임워크

(1) ArrayList: 원소들이 리스트에 삽입된 순서대로 정렬이 됩니다.
파일에 곡 목록이 들어가 있고, 한줄에 한 곡씩에 대한 정보가 들어가 있는 정보를 ArrayList의 sort() 메소드를 사용하면 정렬이 됩니다.

```
ArrayList<JCheckBox> checkboxList;
ArrayList<Integer> trackList = null;
```

서버로부터 클라언트에 전송할 outputStream을 담을 ArrayList입니다.

```
private ArrayList<ObjectOutputStream> outputStreamList;
```

(2) HashMap: 원소들을 이름/값 쌍 형식으로 저장하고 접근할 수 있게 해줍니다.
otherSeqsMap이라 하는 HashMap변수에 String/Boolean의 형식으로 저장합니다.

```
HashMap<String, Boolean[]> otherSeqsMap = new HashMap<String, Boolean[]>{};
```

(3) Vector: 사용자가 배열을 사용할 때 배열의 크기를 벗어나는 인덱스에 접근하면 java.lang.ArrayIndexOutOfBoundsException이 발생하기 때문에 배열을 사용할 때는 충분한 크기로 설정해야 합니다. 그러나 사용할 배열의 크기를 미리 예측하는 것은 쉬운 일이 아니기 때문에 동적인 길이로 여러 데이터형을 저장할 때 쓰는 Vector클래스를 사용합니다.

```
Vector<String> listVector = new Vector<String>();
```

listVector.add(nameToShow): listVector에 nameToShow 객체를 저장합니다.

3. 기능추가

- 복구하는 버튼을 추가

이는 사용자가 연주를 지속적으로 했을 때 전에 있던 연주를 가지고 오려고 할 때 이 복구 버튼을 누르면 복구가 됩니다.

Actionevent를 사용하여 버튼을 누를 때 이벤트가 발생하게 하였고, FileInputStream과 ObjectInputStream을 사용하여 객체를 저장한 다음에 JCheckBox에서 찍히게 하였습니다.

```
// 비트 박스 패턴 복구하기.
public class MyReadListener implements ActionListener{
    @Override
    public void actionPerformed(ActionEvent e) {
        // 불린의 배열 초기화
        boolean[] checkboxState = null;
        try{
            // FileInputStream 클래스는 바이트 단위의 입력을 받는 클래스 이다.
            FileInputStream fileIn = new FileInputStream(new File("Checkbox.ser"));
            // 파일이나 네트워크를 통해 전달 받은 직열화된 데이터를 다시 원래대로 돌리는 역할을 한다.
            ObjectInputStream is = new ObjectInputStream(fileIn);
            // 파일에서 객체 하나를 읽은 다음 불린 배열로 다시 만든다.
            checkboxState = (boolean[]) is.readObject();
        }catch(Exception ex){
            ex.printStackTrace();
        }
        for (int i = 0; i < 256; i++){
            JCheckBox check = (JCheckBox) checkboxList.get(i);
            // 실제 JCheckBox 객체로 구성된 ArrayList에 들어있는 각각의 체크박스를 원래대로 복수한다.
            if(checkboxState[i]){
                check.setSelected(true);
            }else{
                check.setSelected(false);
            }
        }
        // 현재 연주중인것을 멈춘다.
        sequencer.stop();
        // 체크상자의 새로운 상태를 이용하여 시퀀스를 재구성한다.
        buildTrackAndStart();
    }

}
```

```
// 복구 버튼 생성 & 붙이기
JButton read = new JButton("복구");
read.addActionListener(new MyReadListener());
buttonBox.add(read);
```

- 사용자 이름 받기

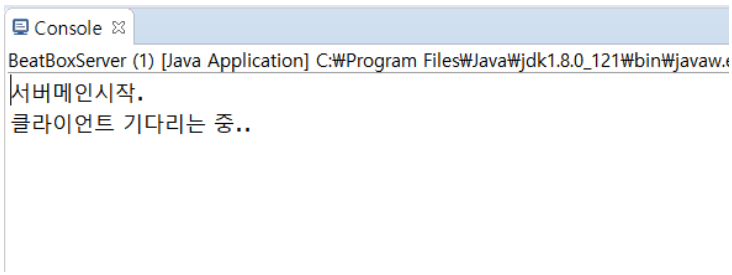
원래의 프로그램은 new BeatBoxClient().startUp(args[0])을 호출하여 사용자의 이름을 받지 않고 바로 프로그램이 실행되나 사용자가 프로그램 중 채팅을 하였을 때 누가 누군지를 알기 위해서 사용자 이름을 받는 Scanner 코드를 추가 시켰고, 두 사람이 채팅 할 적에 누가 누구의 것인지를 모르기 때문에 JFrame에 사용자의 이름을 적어주어 사용자의 편의성을 제공하였습니다.

```
public static void main(String[] args) {
  // new BeatBoxClient().startUp(args[0]);
    System.out.print("사용자명을 입력하세요 : ");
    // 사용자명 입력받기
    Scanner scanner = new Scanner(System.in);
    // 사용자명 읽어오기
    String name = scanner.nextLine();
    // startUp()실행
    new BeatBoxClient().startUp(name);
}
```

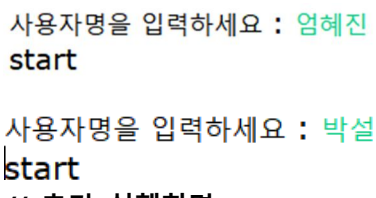
```
    /* 클라이언트의 GUI 생성함수*/
    public void buildGUI(){

        // 프레임을 만든다.
-       theFrame = new JFrame(userName + "님 비트박스");
```

4. 실행화면

// 서버실행화면

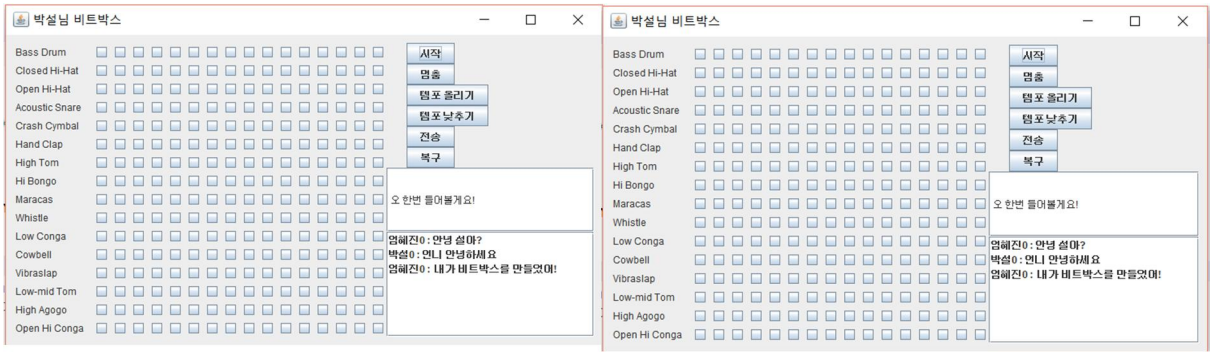


// 클라이언트 실행화면



// 초기 실행화면

// 두 채팅자 간의 입력받는 화면



// 서로의 음악을 받아와 더 추가하여 보내는 화면

5. 소스코드

5.1 서버

```
package finalproject;

import java.io.*; // 입출력과 관련된 기능 제공
import java.net.*; // 네트워크와 관련된 기능 제공하며 각종 프로토콜을 제공
import java.util.*; // 유틸리티 클래스 포함

public class BeatBoxServer {
            //클라이언트한테 보낼 outputStream을 담을 ArrayList
    private ArrayList<ObjectOutputStream> outputStreamList;

    public static void main(String[] args) {
            // "서버메인시작" 메세지 출력
        System.out.println("서버메인시작.");
     // 새로운 BeatBoxServer로 이동
        new BeatBoxServer().go();
    }
    /* 클라이언트로 부터 데이터를 받아오는 스레드 */
    class MyServerThread implements Runnable {
            // 클라이언트로부터 데이터를 불러올 inputStream 생성
        private ObjectInputStream inputStream;
     // 클라이언트 Socket 생성
        Socket clientSocket;

        public MyServerThread(Socket socket) {
            try {
                    // 클라이언트 Socket
                clientSocket = socket;
             // 클라이언트로부터 불러온 데이터 inputStream에 저장
                inputStream = new ObjectInputStream(clientSocket.getInputStream());
             // 서버 inputStream 생성 메세지 출력
                System.out.println("서버 인풋스트림생성");
            } catch (IOException e) {
                e.printStackTrace();
            } // 예외처리
        } // 생성자 끝
        @Override
        public void run(){
           // Object o1 생성
            Object o1 = null;
         // Object o2 생성
            Object o2 = null;
            try{
                while((o1 = inputStream.readObject())!=null){
                     // Object o2에 접근한 객체를 readObject() 사용하여 Object로 반환
                    o2 = inputStream.readObject();
                    // read two object 메세지 출력
```

```
                    System.out.println("read two objects");
                    tellEveryOne(o1, o2);
                }
            }catch(Exception e){
                e.printStackTrace();
            } //예외처리
        } // run 끝
    } // 내부 클래스 끝
    public void go(){
        // 클라이언트에게 보낼 outputStreamList 생성
        outputStreamList = new ArrayList<ObjectOutputStream>();

        try {
            // 서버 Socket 포트 번호 4242로 설정
            ServerSocket serverSocket = new ServerSocket(4242);

            while(true){
                    // 대기 상태 메세지 출력
                System.out.print("클라이언트 기다리는 중.. ");
                // 서버에서 클라이언트와 통신하기 위한 새로운 Socket을 만든다.
                Socket clntSocket = serverSocket.accept();
                                    // 클라이언트 접속 수락, 접속 성공 메세지 출력
                System.out.println(clntSocket.getInetAddress()+"에서 접속하였습니다. ");
                // 새로운 클라이언트 outputStream 생성
                ObjectOutputStream outputStream = new ObjectOutputStream(clntSocket.getOutputStream());
                // 리스트에 outputStream 추가
                outputStreamList.add(outputStream);
                // 새로운 클라이언트 Socket 스레드 생성
                Thread myServerThread = new Thread(new MyServerThread(clntSocket));
                // 스레드 시작
                myServerThread.start();
            }
        } catch (Exception e) {
            e.printStackTrace();
        } // 예외처리
    } // go 메소드 끝

    public void tellEveryOne(Object one, Object two){
        // 컬렉션 클래스인 outputStreamList를 읽기 위한 iterator 생성
        Iterator iterator = outputStreamList.iterator();
        // 변수에 다음 데이터가 있을 때까지 출력
        while(iterator.hasNext()){
            try{
                    // 데이터가 없을 때까지 계속 반복
                ObjectOutputStream outputStream = (ObjectOutputStream)iterator.next();
                // outputStream 객체에 Object 형식으로 적는다
                outputStream.writeObject(one);
                // outputStream 객체에 Object 형식으로 적는다
                outputStream.writeObject(two);
```

```
                }catch(Exception e){
                    e.printStackTrace();
                } // 예외처리
            }
        } // tellEveyone 메소드 끝
    } // 클래스 끝
```

5.2 클라이언트

```
package finalproject;

import java.awt.*; // 사용자 인터페이스의 작성 및 그래픽과 이미지의 모든 클래스 포함
import javax.swing.*; // awt를 기반으로 나온 패키지
import javax.swing.event.*; // Swing 컴퍼넌트에 의해 트리거되는 이벤트를 제공
import javax.sound.midi.*; // 음악 재생과 관련된 패키지
import java.util.*; // 유틸리티 클래스가 포함
import java.awt.event.*; // 위의 awt 컴포넌트의 이벤트를 제어하는 패키지
import java.io.*; // 입추력과 관련된 기능 제공
import java.net.*; // 네트워크와 관련된 기능을 제공하며 각종 프로토콜을 제공

public class BeatBoxClient {
    // 창을 만드는 JFrame객체
    JFrame theFrame;
    // 보조프레임 생성
    JPanel mainPanel;
```

```
    // List프레임 생성
    JList incomingList;
    // Text프레임 생성
    JTextField userMessage;
    // 체크 상자를 ArrayList에 저장한다.
    ArrayList<JCheckBox> checkboxList;
    int nextNum;
    // 객체에 대한 참조값을 저장하는 배열 생성
    // 배열과 다른 점은 하나의 Vector에 저장될 수 있고, 길이도 필요에 따라 증감 할 수 있다는 점이 배열과 다르다.
    Vector<String> listVector = new Vector<String>();
    String userName;
    // ObjectInputStream 생성
    ObjectInputStream in;
    // ObjectOutputStream 생성
    ObjectOutputStream out;
    // HashMap생성
    HashMap<String, boolean[]> otherSeqsMap = new HashMap<String, boolean[]>();
    // 시퀀서 생성
    Sequencer sequencer;
    // 시퀀스 생성
    Sequence sequence;
    // mySequence 초기화
    Sequence mySequence = null;
    Track track;
    // GUI레이블을 만들 때 사용할 악기명을 String 배열로 저장한다.
    String[] instrumentNames = {"Bass Drum", "Closed Hi-Hat", "Open Hi-Hat", "Acoustic Snare",
            "Crash Cymbal", "Hand Clap", "High Tom", "Hi Bongo", "Maracas", "Whistle", "Low Conga",
            "Cowbell", "Vibraslap", "Low-mid Tom", "High Agogo", "Open Hi Conga"
    };
    //이 채널은 피아노의 각 건반이 서로 다른 드럼으 ㄹ나ㅏㅌ내는 것과 같다고 보면된다.
    int[] instruments = {35, 42, 46, 38, 49, 39, 50, 60, 70, 72, 64, 56, 58, 47, 67, 63};

public static void main(String[] args) {
        // new BeatBoxClient().startUp(args[0]);
            System.out.print("사용자명을 입력하세요 : ");
            // 사용자명 입력받기
          Scanner scanner = new Scanner(System.in);
        // 사용자명 읽어오기
          String name = scanner.nextLine();
        // startUp()실행
          new BeatBoxClient().startUp(name);
    }

    /* 서버연결 함수 */
    public void startUp(String name){
        userName = name;
        try{
            System.out.println("start");

            // 서버의 IP주소와 TCP 포트번호
            // 서버에 Socket 연결을 한다.
```

```
            Socket sock = new Socket("127.0.0.1", 4242);
          // 역직렬화 : 객체 복구
          // 객체를 읽는다.
            in = new ObjectInputStream(sock.getInputStream());
          // 직렬화된 객체 저장
            out = new ObjectOutputStream(sock.getOutputStream());
          // 스레드 생성, Runnable 객체의 인스턴스를 만든다.
            Thread remote = new Thread(new RemoteReader());
          // 스레드 시작
            remote.start();
        // 예외 잡기
        } catch (IOException e) {
            System.out.println("서버에 연결할 수 없습니다.");
            e.printStackTrace();
        }
       // setUpMidi()와 buildGUI() 호출
        setUpMidi();
        buildGUI();
}

/* 클라이언트의 GUI 생성함수*/
 public void buildGUI(){

        // 프레임을 만든다.
        theFrame = new JFrame(username + "님 비트박스");
       // 프레임의 기본 레이아웃 관리자 생성
        BorderLayout layout = new BorderLayout();
       // 패널 생성
        JPanel background = new JPanel(layout);
       // 비어있응 경계선을 사용하여 패널 둘레와 구성요소가 들어가는 자리 사이에 빈 공간을 만들 수 있다.
        background.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

       // 체크 상자를 ArrayList에 저장한다.
        checkboxList = new ArrayList<JCheckBox>();

        //Y축으로 정렬!
       // BoxLayout는 구성요소를 수직방향으로 쌓을 수 있다.
        Box buttonBox = new Box(BoxLayout.Y_AXIS);

       // 시작버튼 생성 & 붙이기
        JButton start = new JButton("시작");
        start.addActionListener(new MyStartListener());
        buttonBox.add(start);

       // 멈춤 버튼 생성 & 붙이기
        JButton stop = new JButton("멈춤");
        stop.addActionListener(new MyStopListener());
        buttonBox.add(stop);

       // 템포올리는 버튼 생성 & 붙이기
        JButton upTempo = new JButton("템포 올리기");
```

```
  upTempo.addActionListener(new MyUpTempoListener());
  buttonBox.add(upTempo);

// 템포 낮추는 버튼 생성 & 붙이기
  JButton downTempo = new JButton("템포 낮추기");
  downTempo.addActionListener(new MyDownTempoListener());
  buttonBox.add(downTempo);
// 전송 버튼 생성 & 붙이기
  JButton send = new JButton("전송");
  send.addActionListener(new MySendListener());
  buttonBox.add(send);
// 복구 버튼 생성 & 붙이기
  JButton read = new JButton("복구");
  read.addActionListener(new MyReadListener());
  buttonBox.add(read);

  // 창을 닫을 수 있게 설정
  theFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// 유저가 메시지를 쓸수있게 텍스트 필드 생성 & 붙이기
  userMessage = new JTextField();
  buttonBox.add(userMessage);

 // JList를 통해 받는 메시지가 화면에 표시된다.
// JList 생성
  incomingList = new JList();
// addListSelectionListener 리스너
  incomingList.addListSelectionListener(new MyListSelectionListener());
  incomingList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
// JScrollerPane을 만든다.
  JScrollPane theList = new JScrollPane(incomingList);
// buttonBox에 스크롤 기능을 추가한다.
  buttonBox.add(theList);
// 처음에는 데이터가 없다.
  incomingList.setListData(listVector);
//박스 클래스 생성, 이 박스는 y축 정렬
  Box nameBox = new Box(BoxLayout.Y_AXIS);
// 16개의 악기들을 붙인다.
  for(int i =0 ; i<16 ; i++){
        nameBox.add(new Label(instrumentNames[i]));
  }

// 오른쪽에 buttonBox를 붙인다.
  background.add(BorderLayout.EAST, buttonBox);
// 왼쪽에 nameBox를 붙인다.
  background.add(BorderLayout.WEST, nameBox);
// 프레임에 패널 붙이기.
  theFrame.getContentPane().add(background);

//GridLayout 생성
  GridLayout grid= new GridLayout(16,16);
```

```
        //컴포넌트 사이 간격 조정
        grid.setVgap(1);
        grid.setHgap(2);
       // JPanel 생성
        mainPanel = new JPanel(grid);
       // 패널의 가운데에 메인 페널을 붙인다.
        background.add(BorderLayout.CENTER, mainPanel);
       //체크박스를 만들고 모든값을 체크되지않은 상태로 ArrayList 와 GUI패널에 추가
        for(int i =0 ; i<256 ; i++){
              JCheckBox c = new JCheckBox();
              c.setSelected(false);
              checkboxList.add(c);
              mainPanel.add(c);
        } // for문 끝
       // 프레임의 크기 및 위치 지정
        theFrame.setBounds(50, 50, 300, 300);
       //자동으로 사이즈를 알맞게 맞춰서 나오게 한다.
        theFrame.pack();
       // 프레임을 보이게 하라.
        theFrame.setVisible(true);
  } // 메소드 끝
 // 시퀀서, 시퀀스, 트랙을 만들기 위한 일반적인 미디 관련 코드.
  public void setUpMidi(){
        try{
           // sequencer 생성
           // try/catch 블록으로 감싸놓았기 때문에 getSequencer()를 호출해도 문제가 생기지 않는다.
              sequencer = MidiSystem.getSequencer();
            // sequencer 열기
              sequencer.open();
            // 시퀀스 만들기
              sequence = new Sequence(Sequence.PPQ, 4);
            // 트랙 만들기
              track = sequence.createTrack();
                                    sequencer.setTempoInBPM(120);
       // 예외 처리를 한다.
        }catch (Exception e){
              e.printStackTrace();
        }
  } // 메소드 끝

 // 가장 중요한 부분 Start 버튼을 누르면 실행되는 함수
  public void buildTrackAndStart(){
           // 각 악기의 열여섯 박자에 대한 값을 원소가 16개인 배열에 저장한다.
           // 어떤 악기가 특정 박자에서 연주되어야 하면 그 원소의 값에 건반 번호를 넣고 그 반대라면 0을 집어넣는다.
      ArrayList<Integer> trackList = null;
     //기존 트랙을 지우고 새로 만든다.
        sequence.deleteTrack(track);
       // Sequence에서 새로운 Track을 가져온다.
        track =sequence.createTrack();

       // 열 16개 모두에 대해 같은 작업을 처리합니다.
```

```
        for(int i =0; i<16;i++){
            // ArrayList생성
            trackList = new ArrayList<Integer>();
          //체크가 되어있다면 trackList에 해당 악기의 숫자를 넣고 안되있다면 0
            for(int j=0; j<16;j++){
                JCheckBox jc = (JCheckBox) checkboxList.get(j+(16*i));
                if( jc.isSelected()){
                    // 악기 를 key에 넣는다.
                    int key = instruments[i];
                  // 배열에 key를 넣는다.
                    trackList.add(new Integer(key));
                } else {
                    // 그게 아니라면 null값을 넣는다.
                    trackList.add(null);
                }
            } // 내부 순환문 끝
           // 이 악기의 16개의 모든 박자에 대해 이벤트를 만들고 트랙에 추가한다.
            makeTracks(trackList);
        }// 외부 순환문 끝
       // 16번째 박자에는 반드시 이벤트가 있어야 한다.
       // 이렇게 하지 않으면 다시 시작하기 전에 16박자가 모두 끝나지 않을 수 도 있다.
       track.add(makeEvent(192, 9, 1, 0, 15));
        try{
            sequencer.setSequence(sequence);
           // 루프 반복 횟수를 지정하기 위한 메소드.
           // 계속 반복할 수 있도록 sequencer.LOOP_CONTINUOUSLY를 인자로 전달하였다.
            sequencer.setLoopCount(sequencer.LOOP_CONTINUOUSLY);
           // 연주 시작
            sequencer.start();
            sequencer.setTempoInBPM(120);
        }catch(Exception e){
            e.printStackTrace();
        }
    } // 메소드 끝
   // 첫번째 내부 클래스. 버튼의 리스너 역할을 한다.
    public class MyStartListener implements ActionListener {
        public void actionPerformed(ActionEvent a){
            buildTrackAndStart();
        }
    }
   // 두번째 내부 클래스 버튼의 리스너 역할을 한다.
    public class MyStopListener implements ActionListener {
        public void actionPerformed(ActionEvent a){
            sequencer.stop();
        }
    }

    public class MyUpTempoListener implements ActionListener {
        public void actionPerformed(ActionEvent a){
            float tempoFactor = sequencer.getTempoFactor();
           // setTempoFactor()메소드는 시퀀서의 빠르기를 주어진 배율을 가지고 변경한다.
```

```
            // 기본값은 1.0이고, 여기에서는 빠르기를 3% 증가하였다.
              sequencer.setTempoFactor((float) (tempoFactor *1.03));
          }
      }
      public class MyDownTempoListener implements ActionListener {
          public void actionPerformed(ActionEvent a){
              float tempoFactor = sequencer.getTempoFactor();
              // setTempoFactor()메소드는 시퀀서의 빠르기를 주어진 배율을 가지고 변경한다.
             // 기본값은 1.0이고, 여기에서는 빠르기를 3% 감소하였다.
              sequencer.setTempoFactor((float) (tempoFactor *0.97));
          }
      }

// 내부 클래스입니다.
      public class MySendListener implements ActionListener {
            // 사용자가 버튼을 클릭해서 ActionEvent가 발생된 경우에 실행한다.
          public void actionPerformed(ActionEvent a) {
            // 각 체크박스의 상태를 담아두기 위한 부울 배열을 만듭니다.
              boolean[] checkboxState = new boolean[256];
              // checkboxList(체크상자로 이루어진 ArrayList)를 훑어보면서 각 체크상자의 상태를 확인하고 그 결과를 부울 배열에 추가한다.
              for(int i=0;i<256;i++){
                  JCheckBox check = (JCheckBox) checkboxList.get(i);
                  // 선택되어있는 것이 있느냐?
                  if(check.isSelected()){
                      // 그렇다면 그것에 true로 나줘
                      checkboxState[i]=true;
                  } // if문 끝
              } // for 문 끝

              try {
                      // 파일 저장하기. 부울 배열을 직렬화해서 저장하면 된다.
                      // fileoutputstream 객체를 만든다.
                      FileOutputStream fileStream = new FileOutputStream(new File ("Checkbox.ser"));
                      // 파일에 직접연결할 수 없어 보조 객체를 만든다.
                      ObjectOutputStream os = new ObjectOutputStream(fileStream);
                      // 객체를 저장한다.
                      os.writeObject(checkboxState);
                      // 텍스트를 받아와 적는다.
                  out.writeObject(userName + nextNum + " : " + userMessage.getText());
                // 텍스트 객체 저장.
                  out.writeObject(checkboxState);
              } catch (Exception e) {
                   System.out.println("미안. 서버에서 받지 못했어..");
              } // try/catch 끝
              userMessage.setText(" ");
          } // 메소드 끝
      } // 내부클래스 끝
      // 내부 클래스
      // 비트 박스 패턴 복구하기.
      public class MyReadListener implements ActionListener{
                  @Override
```

```
                public void actionPerformed(ActionEvent e) {
                        // 불린의 배열 초기화
                        boolean[] checkboxState = null;
                        try{
                                // FileInputStream 클래스는 바이트 단위의 입력을 받는 클래스 이다.
                                FileInputStream fileIn = new FileInputStream(new File("Checkbox.ser"));
                                // 파일이나 네트워크를 통해 전달 받은 직열화된 데이터를 다시 원래대로 돌리는 역할을 한다.
                                ObjectInputStream is = new ObjectInputStream(fileIn);
                                // 파일에서 객체 하나를 읽은 다음 불린 배열로 다시 만든다.
                                checkboxState = (boolean[]) is.readObject();
                        }catch(Exception ex){
                                ex.printStackTrace();
                        }
                        for (int i = 0; i < 256; i++){
                                JCheckBox check = (JCheckBox) checkboxList.get(i);
                                // 실제 JCheckBox 객체로 구성된 ArrayList에 들어있는 각각의 체크박스를 원래대로 복수한다.
                                if(checkboxState[i]){
                                        check.setSelected(true);
                                }else{
                                        check.setSelected(false);
                                }
                        }
                        // 현재 연주중인것을 멈춘다.
                        sequencer.stop();
                        // 체크상자의 새로운 상태를 이용하여 시퀀스를 재구성한다.
                        buildTrackAndStart();
                }

        }

        // 사용자가 메시지 목록에서 한 메시지를 선택했을 경우에 실행되는 ListSelectionListener이다.
        // 사용자가 메시지를 선택하면 즉시 그 메시지와 연관된 비트 패턴을 불러오고 연주를 시작합니다.
        public class MyListSelectionListener implements ListSelectionListener{
                public void valueChanged(ListSelectionEvent e) {
                        if(!e.getValueIsAdjusting()){
                                // 사용자가 메시지를 선택하면
                                String selected = (String) incomingList.getSelectedValue();
                                if(selected!=null){
                                        // otherSeqsMap이라하는 HashMap 객체를 불러오고 연주를 시작한다.
                                        boolean[] selectedState = (boolean[]) otherSeqsMap.get(selected);
                                        changeSequence(selectedState);
                                        // 멈춘다.
                                        sequencer.stop();
                                        // buildTrackAndStart()메소드 호출
                                        buildTrackAndStart();
                                } // if 문 끝
                        }// if문 끝
                }// 메소드 끝
        } // 내부클래스 끝
```

```
// 스레드에서 처리할 작업이다.
// 서버로부터 데이터를 읽어오고, 직렬화된 객체 두 개가 존재한다.
 public class RemoteReader implements Runnable{
        boolean[] checkboxState = null;
        String nameToShow = null;
        Object obj = null;
        // 스레드에서 실행해야 할 작업이 여기 들어가야한다.
        public void run(){
                try{
                // 메시지가 들어오면 객체 두개를 읽어오고 JList에 항목을 추가하는 2가지 과정을 한다.
                        while((obj = in.readObject()) !=null){
                                String nameToShow = (String) obj;
                                // 객체를 읽는다.
                                checkboxState = (boolean []) in.readObject();
                                otherSeqsMap.put(nameToShow, checkboxState);
                                // 첫번째 과정. 목록 데이터에 대한 Vector 객체를 만들고
                                listVector.add(nameToShow);
                                // JList에 그 Vector를 이용하여 목록에 표시하라는 명령을 한다.
                                incomingList.setListData(listVector);
                        }// while문 끝
                // 예외 처리
                }catch(Exception ex){
                        ex.printStackTrace();
                }
        }// run 끝
} // 내부 클래스 끝

 public class MyPlayMineListener implements ActionListener{
        public void actionPerformed(ActionEvent a){
                if (mySequence != null){
                        sequence  = mySequence; // 내가 만든 시퀀스로 돌아간다.
                }
        } // actionPerformed 끝
} // 내부 클래스 끝

 // 사용자가 목록에서 무엇인가를 선택하면 그 선택된 항목에 해당하는 패턴으로 바로 변경한다.
 public void changeSequence(boolean[] checkboxState){
      for(int i=0 ; i<256 ; i++){
           JCheckBox check = (JCheckBox) checkboxList.get(i);
           if(checkboxState[i]){
                // 만약 checkboxState가 i를 선택하면 true
                check.setSelected(true);
           }else{
                // 아니라면 false
                check.setSelected(false);
           }
      } // for 문 끝
 } // 메소드 끝

// 한 악기의 16박자 전체에 대한 이벤트를 만든다.
```

```
    public void makeTracks (ArrayList list){ //트랙 만들기
            // 반복자를 생성
            Iterator it = list.iterator();

        for(int i=0 ; i<16 ; i++){
            Integer num = (Integer) it.next();
            if(num != null){
                    int numKey = num.intValue();
                    // NOTE ON과 NOTE OFF 이벤트를 만들고 트랙에 추가한다.
                track.add(makeEvent(144,9,numKey,100,i));
                track.add(makeEvent(128,9,numKey,100,i+1));
            }
        } // for 문 끝
    } // 메소드 끝

    // 메시지를 만들기 위한 인자 5개
    public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick){
        MidiEvent event = null;
        try{
          // 메소드 매개변수를 써서 메시지와 이벤트를 만든다.
            ShortMessage a = new ShortMessage();
            a.setMessage(comd, chan, one, two);
            event = new MidiEvent(a, tick);
        }catch (Exception e){}
        // 이벤트를 리턴한다.(메시지가 모두 들어있는 MidiEvent)
        return event;
    } // 메소드 끝
}
```

6. 느낀 점

엄혜진

처음 이 작업을 했을 때는 어떻게 작업을 수행해야 할지 막막함이 들었다.
어느 부분을 어떻게 해결해야 할지, 어떤 기능을 추가 시켜야 할지 많은 생각을 하였지만 교수님께서 말씀하신 헤드퍼스트 책을 보고 그 생각이 180도 바뀌었다. 평소 헤드퍼스트를 할 적에는 별로 중요하지 않다고 생각한 점이 많았었는데 요번에 책을 처음부터 정독을 하면서 다시 읽어본 결과 서버와 클라이언트에 대해서 얕게 알았던 내 지식이 더 깊어졌다는 것을 느꼈고, 프로그램을 추가하는 작업에서도 막힘 없이 술술 풀어나갔다. 좀 더 깊이 해보고 싶었지만 현재에는 과제와 시험으로 인해 여기까지 못한 것에 대한 아쉬움이 있었다.
디자인도 바꿔보고 싶었고, 채팅한 내용을 지우는 작업도 해보고 싶었지만 거기까지 못 나간 것이 아쉬웠고, 방학 때 다시 한번 해봐야겠다는 생각을 하였다. 또한 요번 과제를 통해 스레드 부분이 많이 부족하다는 것을 깨달아서 복습을 더 열심히 해야겠다고 다짐했다.
초반에는 이게 대체 무슨 프로그램인가 의문을 들게 하였지만 과제를 끝낸 지금은 100%는 아니지만 초반보다는 더 깊이 알게 되었고 코드를 보고 이게 어떤 작업인지를 알 수 있어 뿌듯함을 느꼈다.